



Jet Propulsion Laboratory
California Institute of Technology

ESD
IMCE

CAESAR Server

New Design Plan

Maged Elaasar

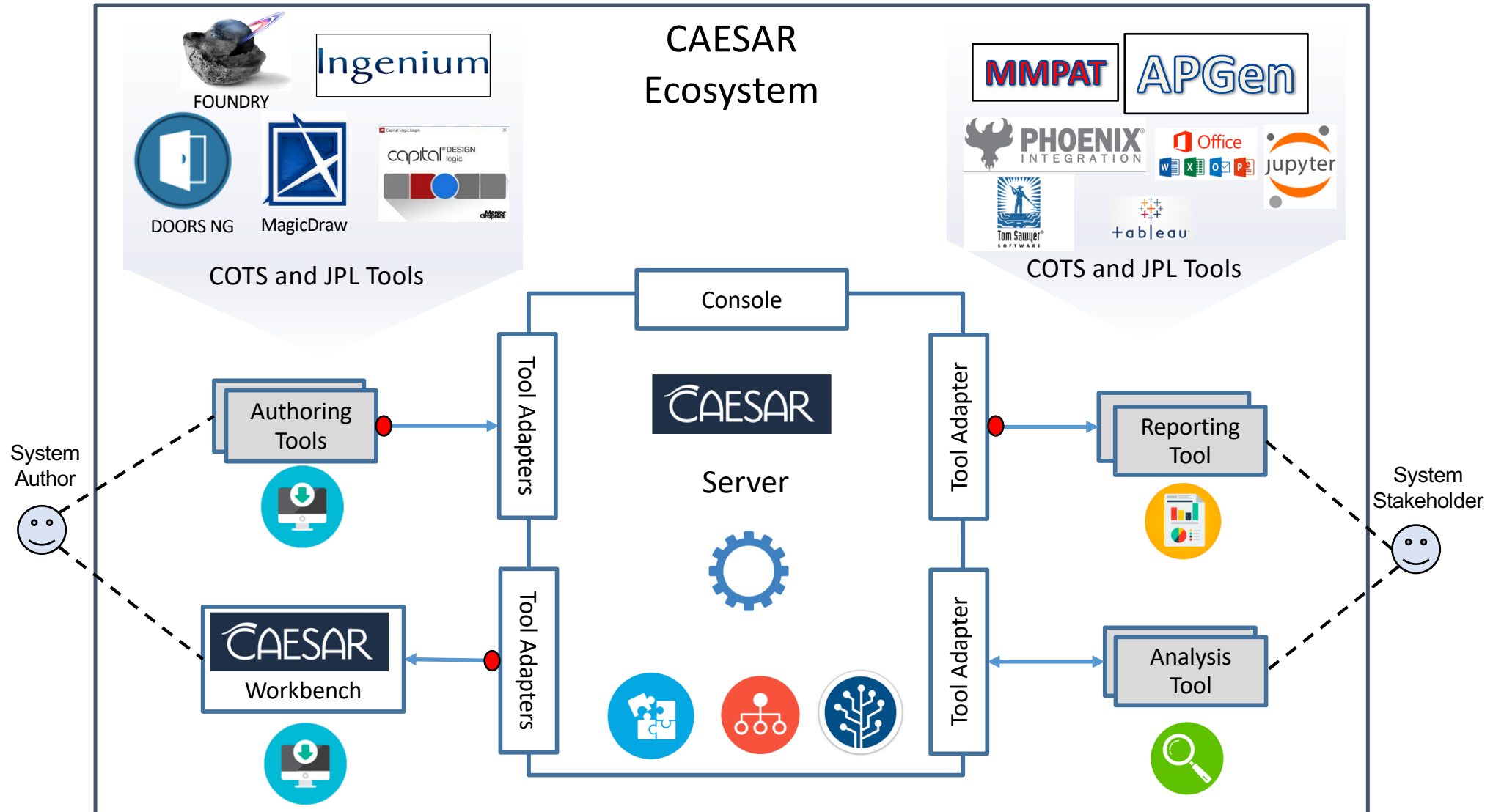
elaasar@jpl.nasa.gov

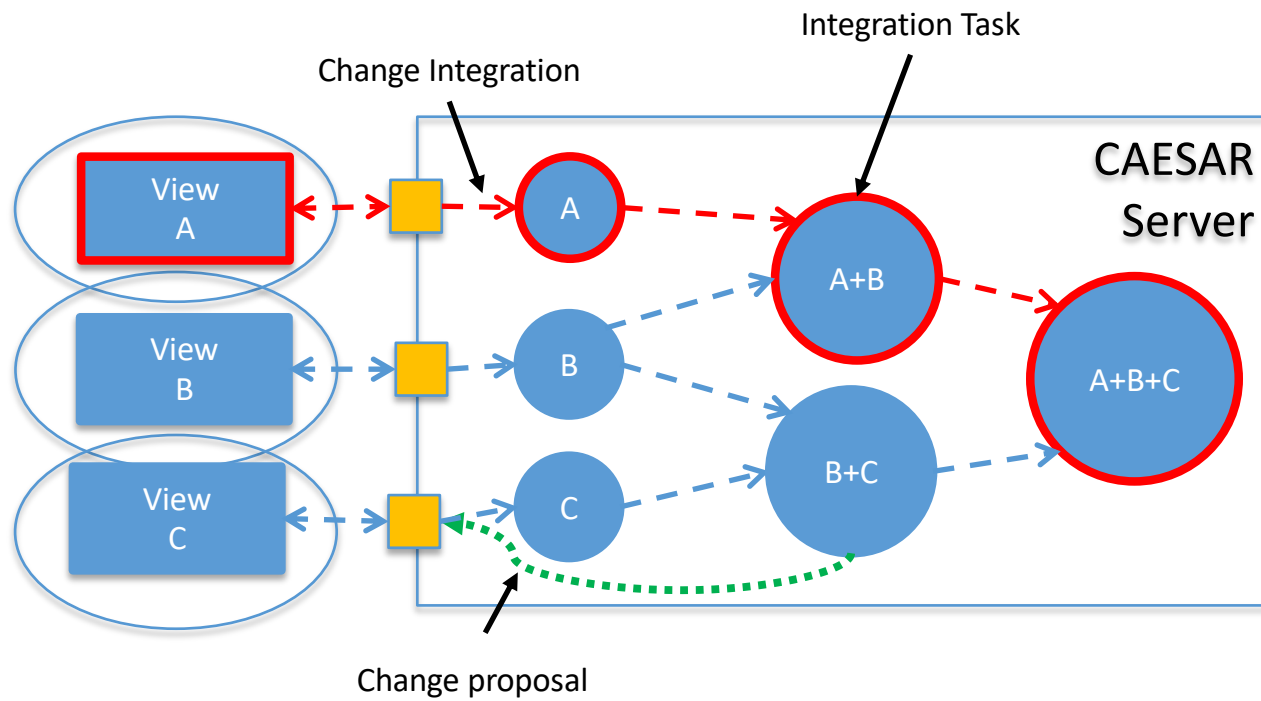
Nicolas Rouquette

nicolas.f.rouquette@jpl.nasa.gov

May 17, 2019

- Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology





- We have implemented the current server and deployed it to two projects
 - It meets 100% of the functional and performance needs for those projects
- We are evolving the design to scale with
 - Number of deployments
 - Number of projects served per deployment
 - Number of users per project
 - Number of disciplines/applications supported
 - Other dimensions that are captured in a demand model (in the backup)
- We are also evolving the design to improve maintainability

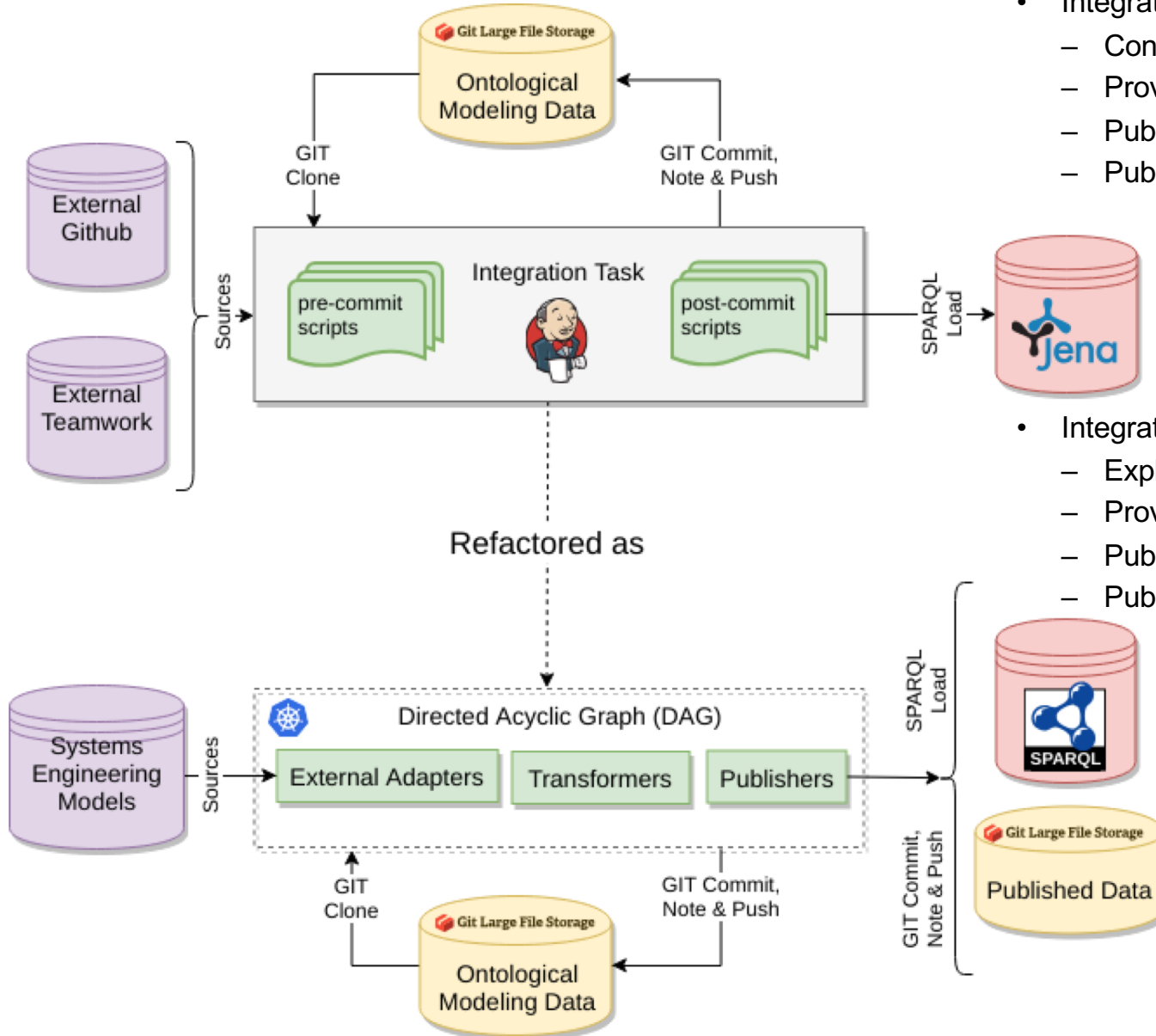
IMCE New Design Strategy

- Step 1: Validate New Design
 - Trades and proof-of concept demo (done)
 - Document and review new design approach (this presentation)
- Step 2: Develop Minimum Viable Product
 - Implement feature parity with current server
 - Deploy to production
 - Retire current server (at the earliest opportunity)
- Step 3: Develop New features
 - This work is to be planned as part of our agile process going forward

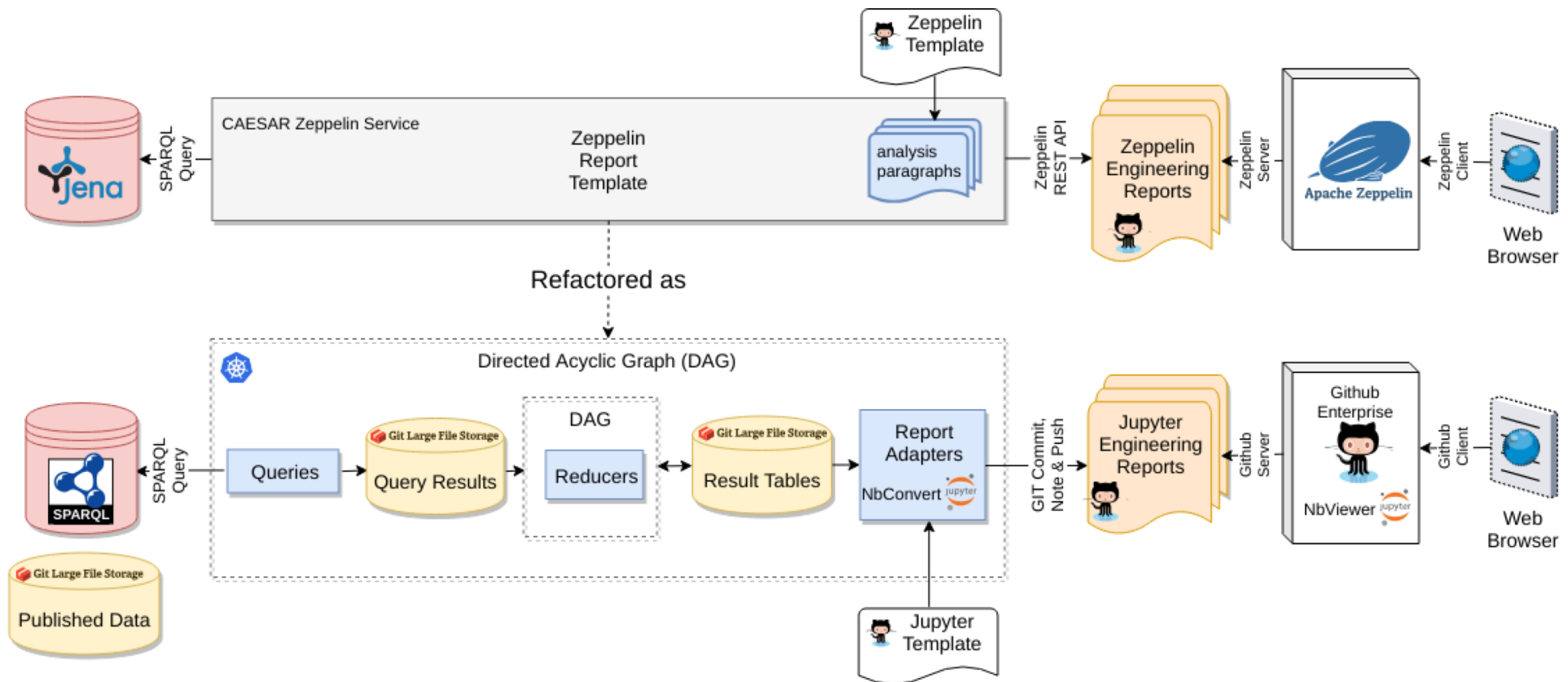


Server Design Changes

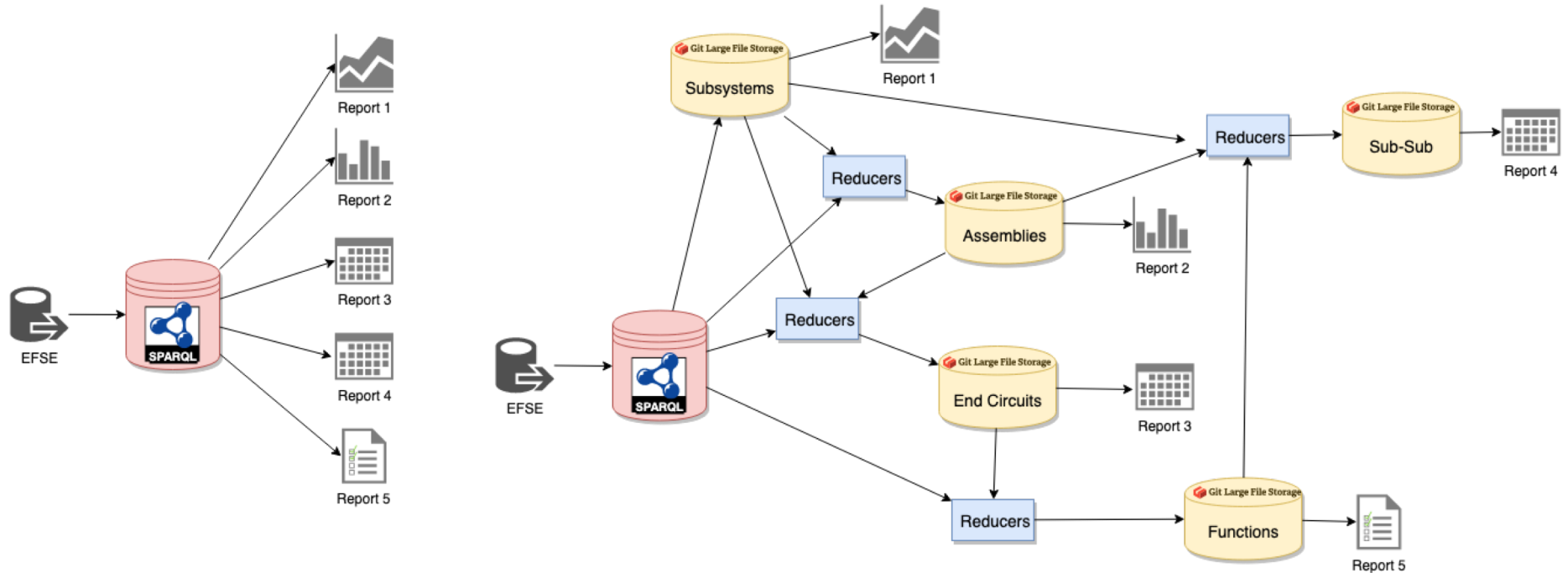
- Functional
 - Integration Workflow
 - Analysis & Reporting
 - Configuration Management
- Under the Hood
 - Service Architecture
 - Service Implementation
 - Service Orchestration
 - Service Deployment
 - Service Security
 - Managed Services



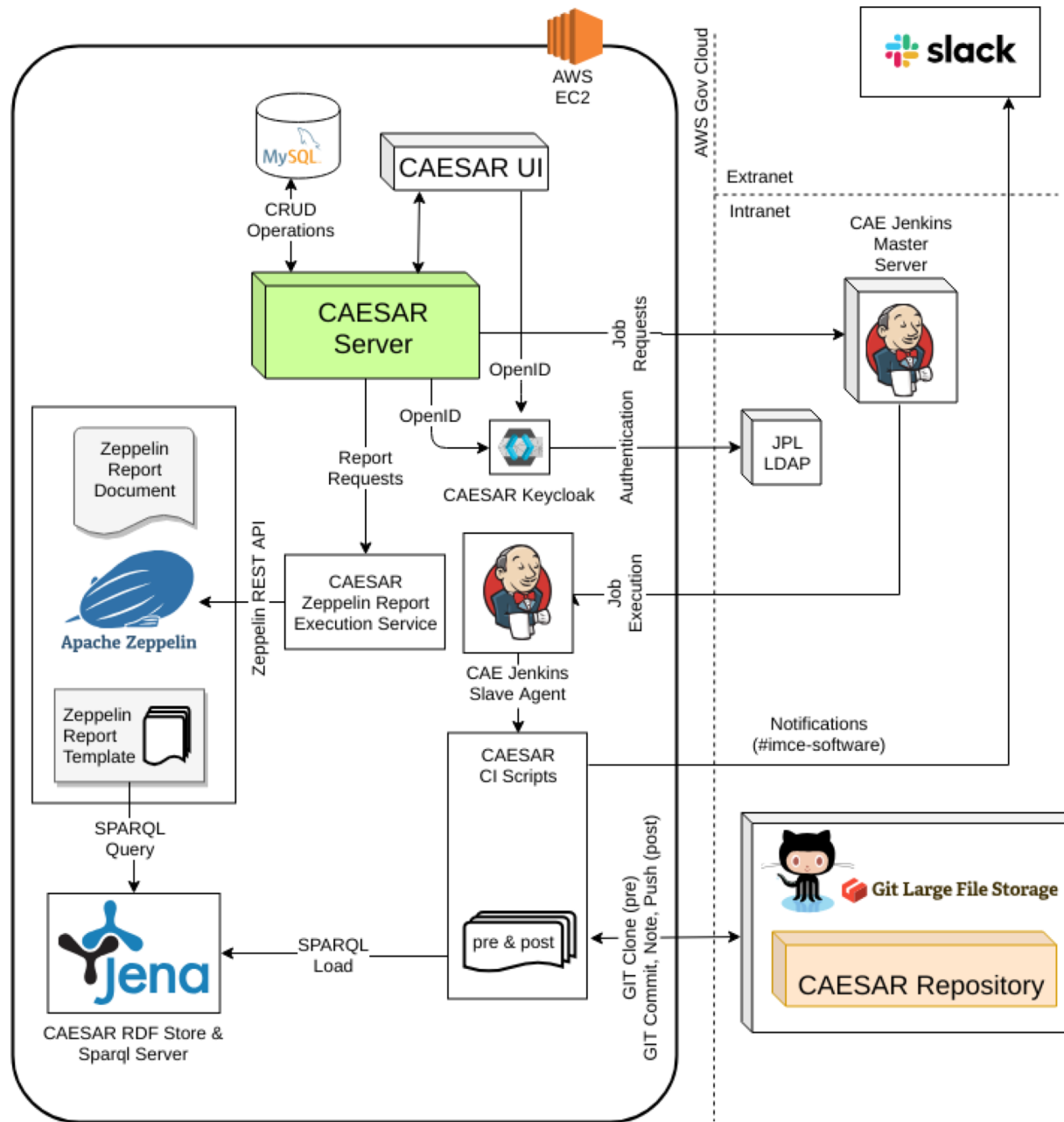
- Integration workflow consists of black-box tasks
 - Configured by (pre-commit, post-commit) scripts
 - Provenance recorded ad-hoc by scripts
 - Published data not configuration managed
 - Published data loaded to Jena repository
- Integration workflow is a DAG of primitive processes
 - Explicit topology of (adapt, transform, publish)
 - Provenance recorded in Git by execution engine
 - Published data C/Med in Git as well
 - Published data loaded to a SPARQL endpoint



- Analysis and reporting concerns are mixed in report template
 - Implicit analysis topology in the data flow in the template
 - Traceability between analysis and report is coarse grained
- Reports are published but not CM'ed
- Reports are viewed through stateful full-fledged servers
- Analysis is specified as a DAG of primitive ops
 - Explicit analysis topology (map, reduce, report)
 - Traceability between analysis and report is fine grained
- Reports are CM'ed in Git
- Reports are viewed with stateless light-weight viewers



- Current workflow is opaque (hidden in report template)
- No provenance available for reports
- Change impact analysis is expensive (error-prone visual comparison of reports)
- New workflow DAG is transparent
- Provenance metadata is stored in Git
- Change impact analysis is fine grained and cheap to calculate



Two deployments:

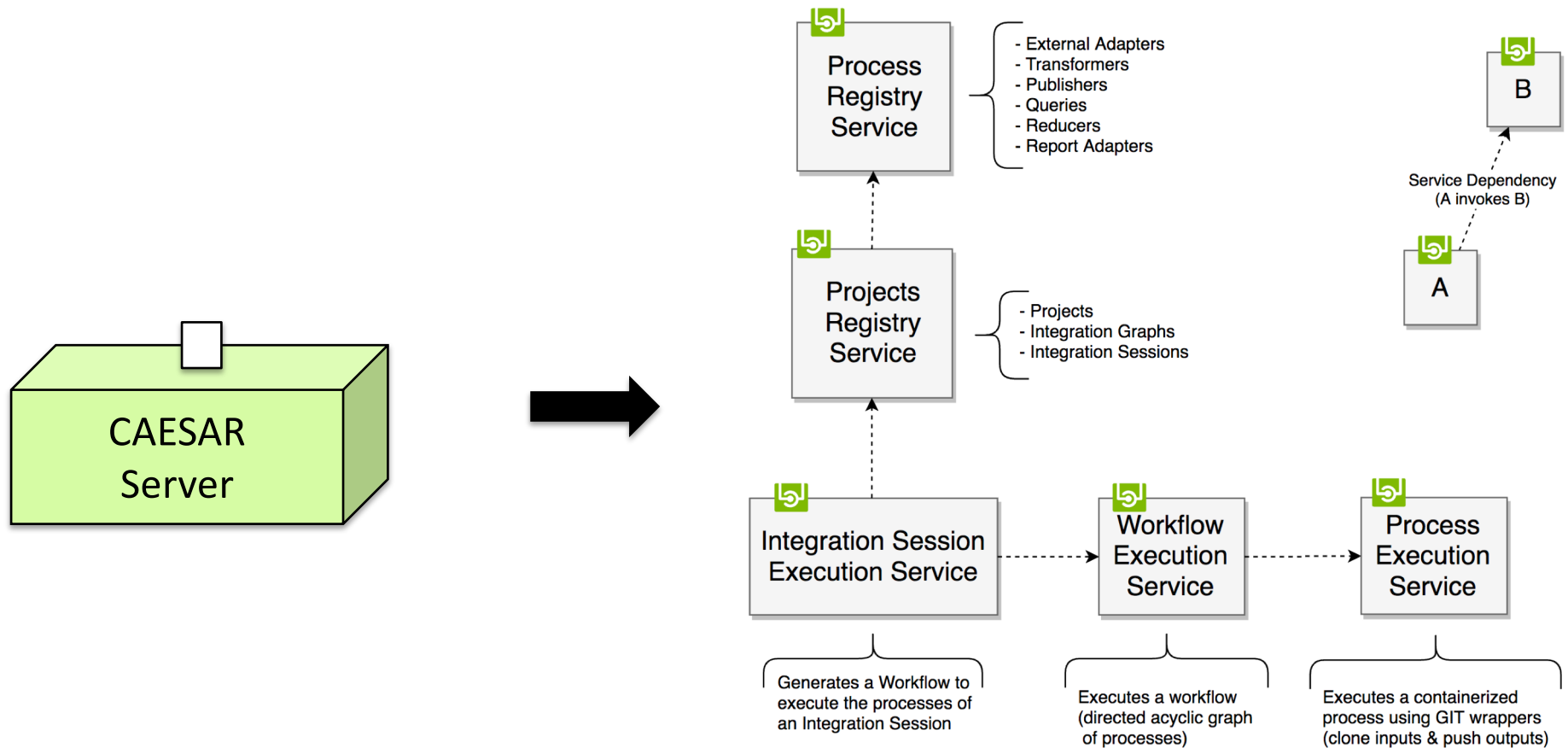
- Production
(EC2 = imce-infr-dev-01)

- Development
(EC2 = imce-infr-dev-02)

Some Statistics:

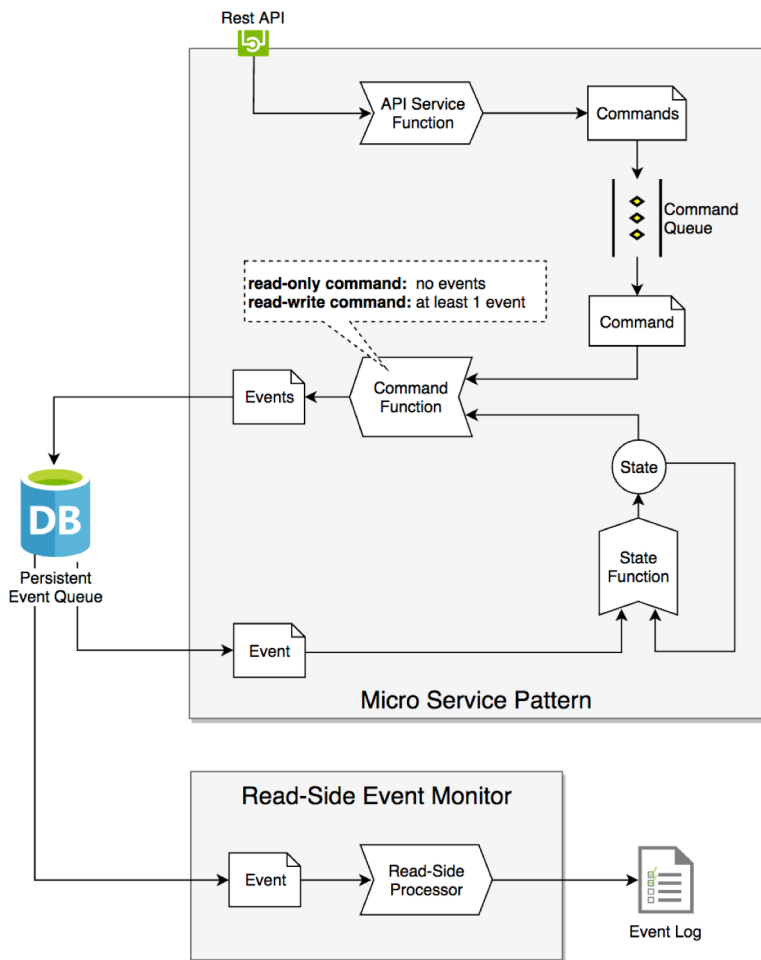
- 5 projects
- > 350 tasks executions
- > 5,000 generated reports

IMCE Service Architecture



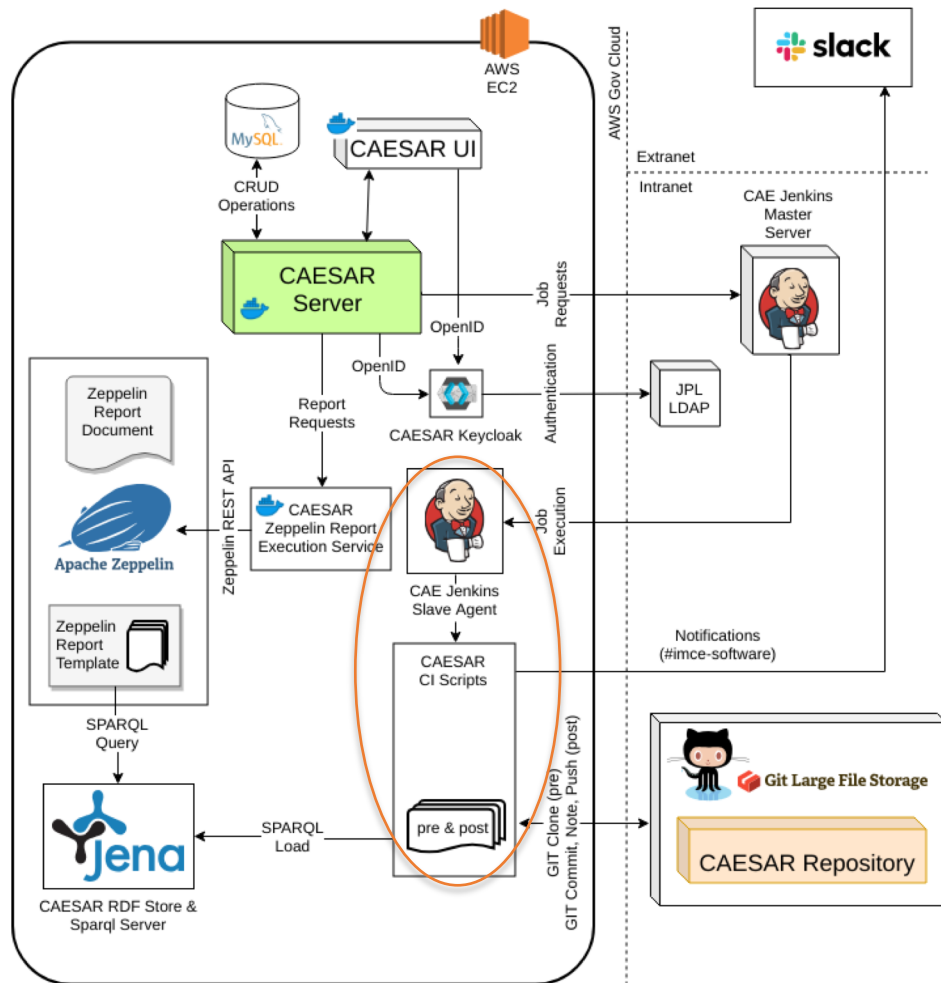
- monolith server architecture

- micro-services architecture using the open source Lagom Framework

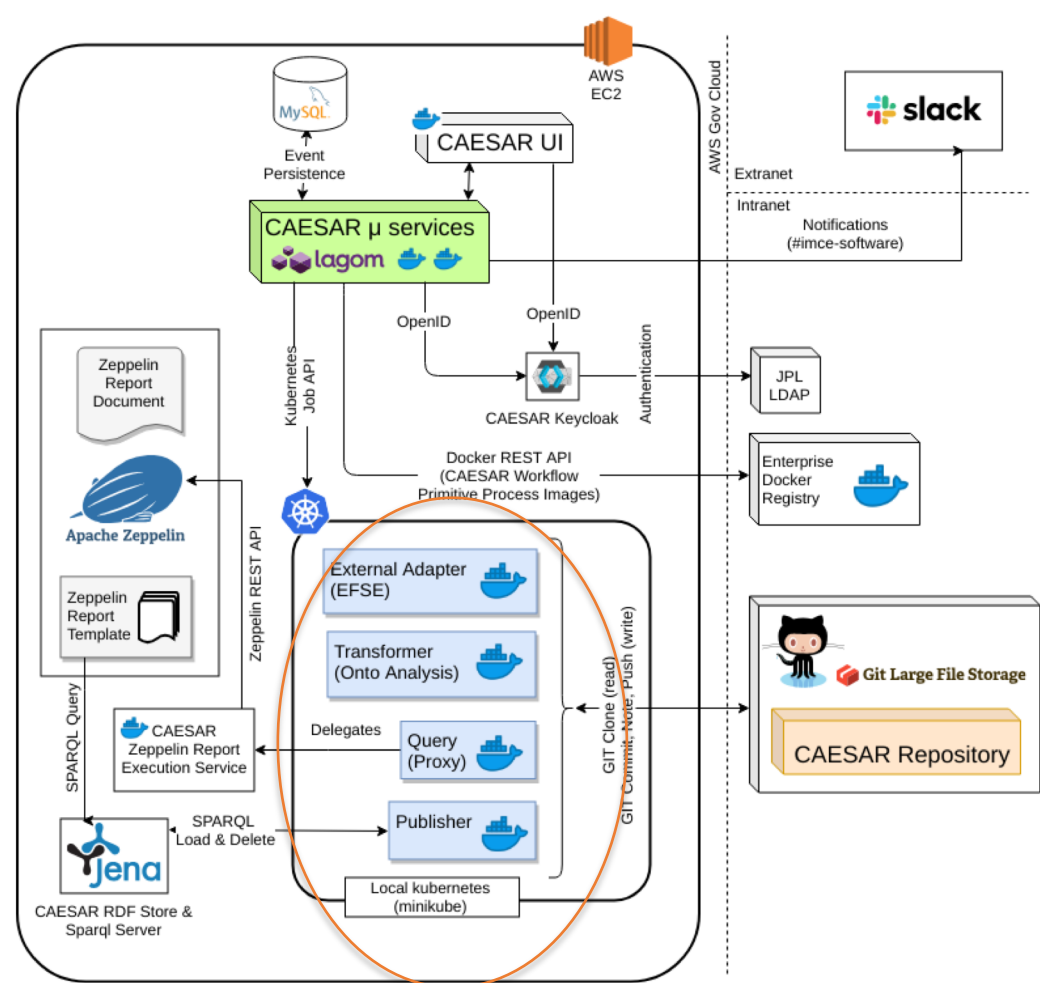


- Current monolith service stores its state in DB
- New micro-services use two patterns:
 - Event Sourcing (ES)
 - application events persist in DB
 - state is **derived** from events
 - Command/Query Responsibility Segregation (CQRS)
 - Distinguish two kinds of operations w.r.t. effects on state
 - A command operation depends on current state and affects it
 - A query operation depends on current state but does not modify it
- ES + CQRS allows separating
 - reads (queries)
 - Query processors can be replicated to handle potentially spikes of demand
 - Great for elasticity!
 - writes (commands)
 - Command processors need to deal with concurrency, transactions, failures,

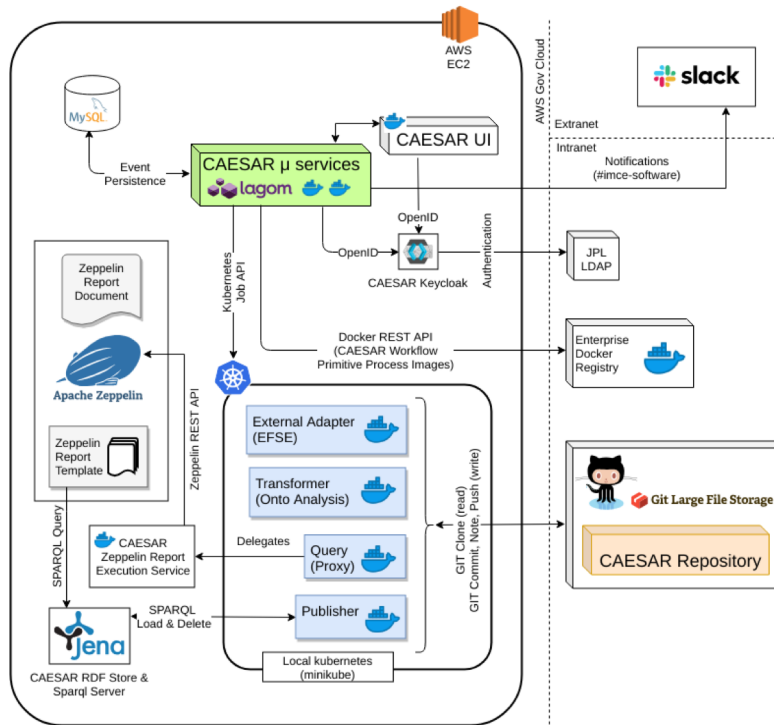
IMCE Service Orchestration



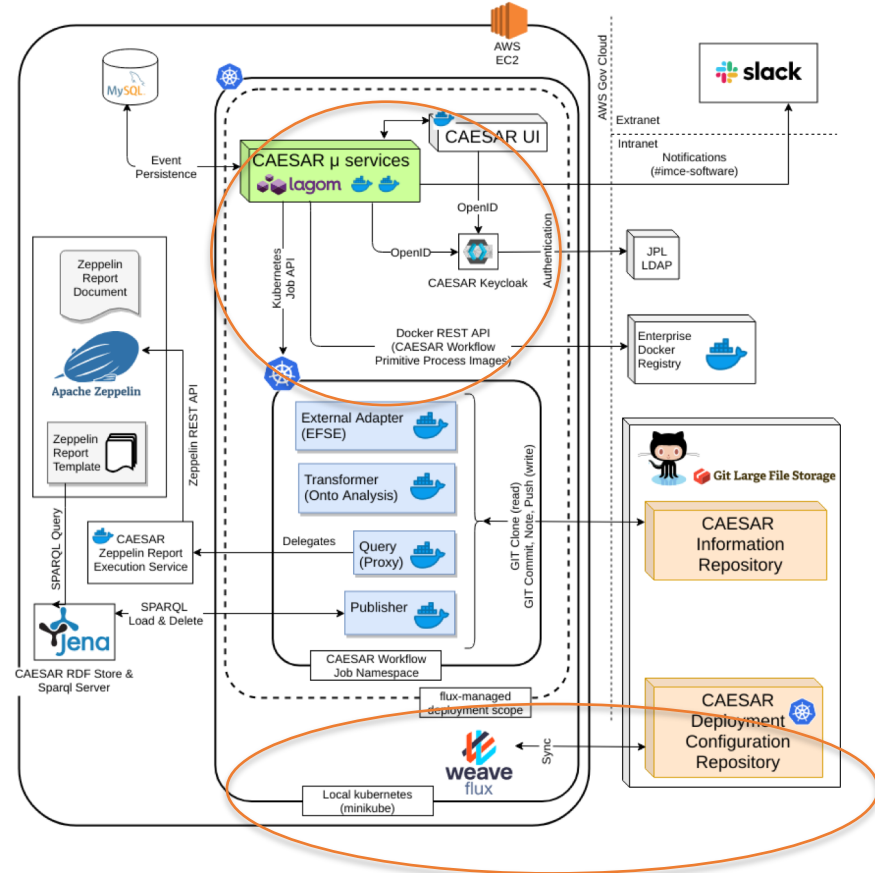
- CAESAR batch Jobs (integration tasks) are orchestrated directly using Jenkins CI



- CAESAR batch jobs (processes) are dockerized and orchestrated with Kubernetes
- Kubernetes helps CAESAR maintain a vendor-neutral strategy with a very high level of interoperability with multiple cloud vendors, including Amazon Web Services

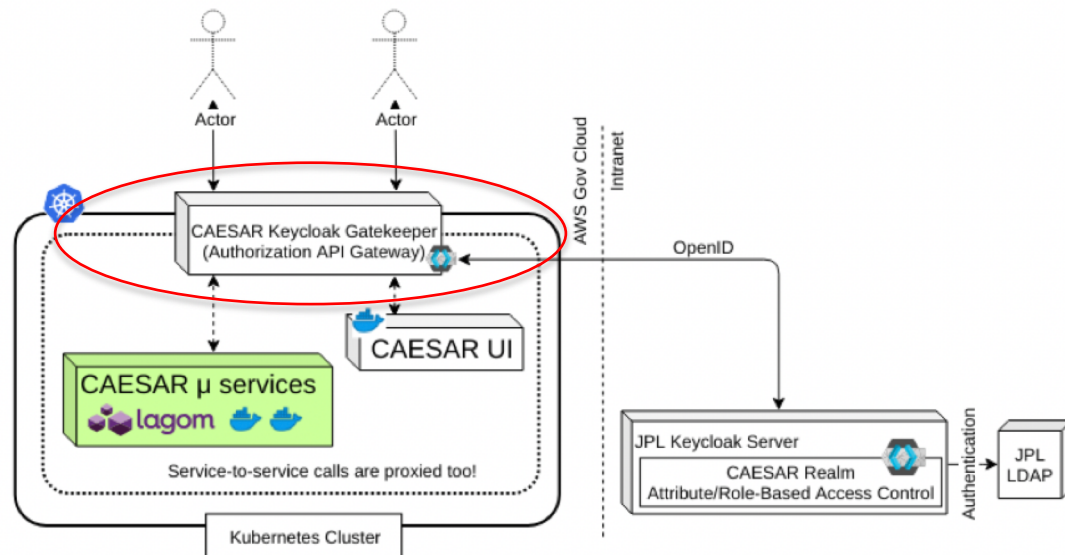
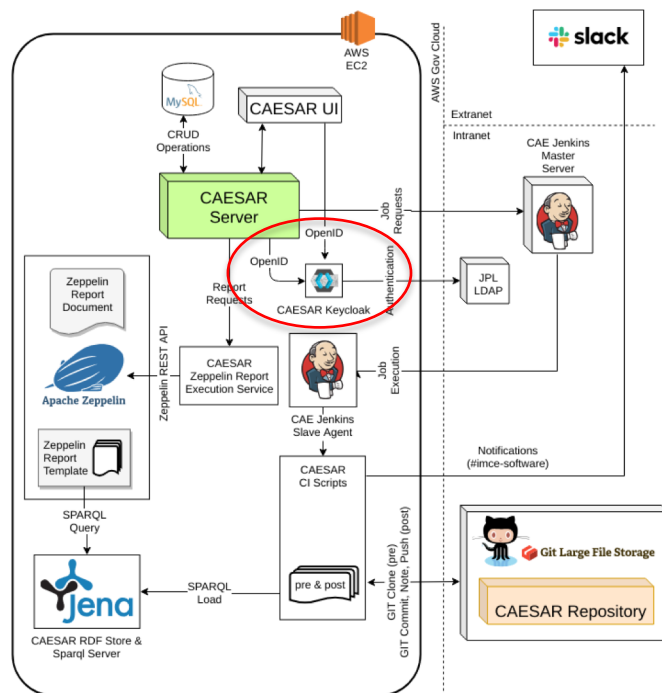


- CAESAR services (and third party dependencies) are deployed manually and on a single node



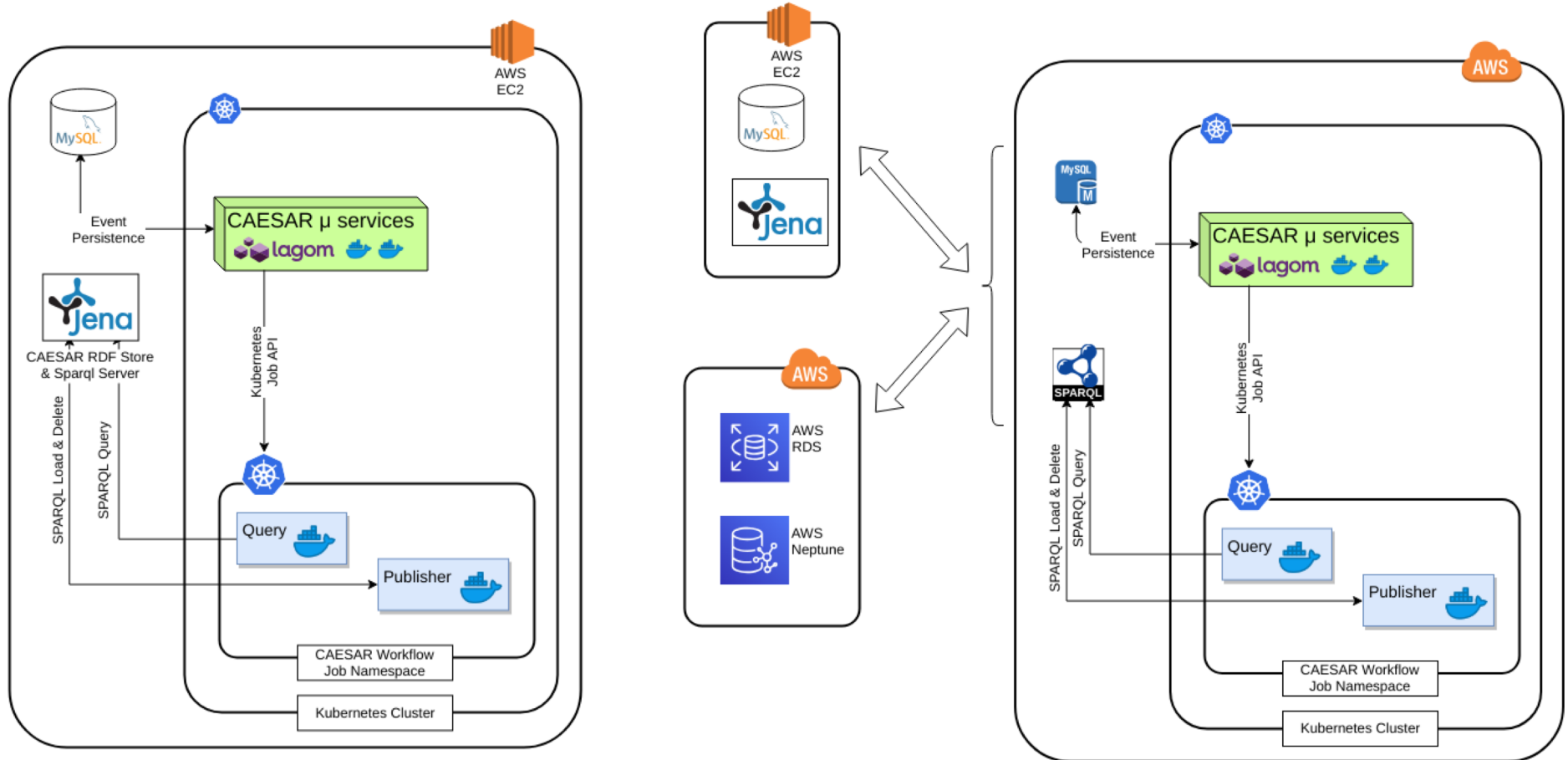
- CAESAR services (and third party dependencies) are dockerized and deployed using Kubernetes on a cluster of nodes (in the cloud or on-premise)
- The deployment is configuration managed in Git and automated using WeaveFlux
 - Easily reproducible CAESAR deployments
 - Facilitate experimenting with alternate Kubernetes environments (OCIO, AWS EKS, ...)
 - Simplify deployment documentation

IMCE Service Security



- OpenID/Connect Authentication with Keycloak
- LDAP as an ID provider
- No authorization
- Individual services handle authentication individually
- Private user data is exposed to CAESAR services (via JWT tokens)

- Project-specific role-based and/or attribute-based authorization via CSAESAR realms in Keycloak
- Authorization API Gateway abstracts Keycloak interface and enforces strict security & access control compliance
- All external & internal APIs are reverse proxied through the Authorization API gateway
- No private user data is exposed to CAESAR services



- New design can leverage cloud-based managed services, ex.:
 - AWS RDS as a relational database
 - AWS Neptune as a SPARQL endpoint
 - AWS EKS as a Kubernetes endpoint
 - AWS EMR as a Jupyter Notebook service